Security metrics at work on the Things in IoT systems *

Chiara Bodei[1 [0000−0002−0586−9333]], Pierpaolo Degano[1 [000−0002−8070−4838]], Gian-Luigi Ferrari[1 [0000−0003−3548−5514]], and Letterio Galletta[2 [0000−0003−0351−9169]]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
{chiara.bodei,pierpaolo.degano,gian-luigi.ferrari}@unipi.it
[2] IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Abstract. The Internet of Things (IoT) is deeply changing our society. Daily we use smart devices that automatically collect, aggregate and exchange data about our lives. These data are often pivotal when they are used e.g. to train learning algorithms, to control cyber-physical systems, and to guide administrators to take crucial decisions. As a consequence, security attacks on devices can cause severe damages on IoT systems that take care of essential services, such as delivering power, water, transport, and so on. The difficulty of preventing intrusions or attacks is magnified by the big amount of devices and components IoT systems are composed of. Therefore, it is crucial to identify the most critical components in a network of devices and to understand their level of vulnerability, so as to detect where it is better to intervene for improving security. In this paper, we start from the modelling language IoT-LySA and from the results of Control Flow Analysis that statically predict the manipulation of data and their possible trajectories. On this basis, we aim at deriving possible graphs of how data move and which are their dependencies. These graphs can be analysed, by exploiting some security metrics - among which those introduced by Barrere, Hankin et al. - offering system administrators different estimates of the security level of their systems.

1 Introduction

Security issues can jeopardise the growth of the Internet of Things (IoT). The risk of intrusions or attacks is magnified by the big amount of devices and components these systems are composed of. Things are less and less material, while the effects of cyber attacks are far from being only virtual. The presence of actuators that act on the physical world makes security particularly crucial. Nevertheless there is some reluctance in adopting security countermeasures because they can impact

* Partially supported by Università di Pisa PRA_2018_66 *DECLWARE: Metodologie dichiarative per la progettazione e il deployment di applicazioni* and by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems).

on costs and performance. As a consequence, techniques able to evaluate where it is more important to intervene for improving security can be useful for IoT designers in choosing which security mechanisms to use.

We intend here to exploit the Security by Design development methodology, introduced in [4,3,9] by taking inspiration from the work by Barrère, Hankin, et al. [1,20], where the authors propose some security metrics for identifying critical components in ICS (Industrial Control System) networks.

Our methodology is based on a kernel of a modelling language IoT-LySA to describe the structure of an IoT system and its interactive capabilities. This language is endowed with a suitable Control Flow Analysis (CFA) that approximates the system by providing an abstract model of behaviour. These abstractions allow predicting how (abstract) data may flow inside the system. In [7] the CFA also includes a data path analysis that supports tracking of the propagation of data, by identifying their possible trajectories among the computing nodes.

In [1], the security level of a system is obtained as the minimum set of CPS (Cyber-Physical System) components that an attacker must compromise in order to impair the functionalities of the system. Moreover, each component is associated to a score, i.e. a value that quantitatively represents the effort (in terms of cost) required by an attacker to compromise it. Scores allow designers to compute the set of components with the minimal cost for the attacker. The metric relies on AND/OR graphs that represent the complex dependencies among network components, and the problem consists in finding a minimal weighted vertex cut in the graph.

Inspired by [1,20], we use a graph model of the CFA results, in particular, we model the data dependency and the logical conditions on which the trigger of a given actuator depends. More precisely, the obtained graph encodes the dependencies of the condition on the raw data collected by sensors and on the logical and aggregation functions applied to the sensor data in processing the decision (a full formalisation of how to derive the graph from the analysis results is left as a future work). For each sub-term of the abstract value, the analysis gives the location where it is introduced and computed and its possible trajectories across the network. By using this information, designers can investigate the security of their system, by reasoning on which is the minimal set of data that can be altered (by tampering the corresponding nodes) in order to affect the capacity of the system to correctly trigger the actuators. In the presence of more choices, by associating score to each node, it is possible to compare the different solutions and establish which are the cheaper ones. The composition of data in the graph can drive the reasoning in a fashion similar to that introduced in [1].

Structure of the paper. The paper is organised as follows. Our approach is introduced in Section 2, with an illustrative example that serves as case study. We briefly recall the process algebra IoT-LySA in Section 3 and the adopted CFA in Section 4. Conclusions are in Section 5. The Appendix completes the formal treatment of the semantics and of the analysis.

2 Motivating example

In this section we illustrate our methodology through a simple scenario concerning the fire alarm and suppression system of an academic department building.

The scenario. Suppose that the building has three floors: (0) at the ground floor there is the library; (1) at the first floor there is the office area; (2) at the second floor there is the IT center and, inside it, the control room of the whole building.

In every room of each floor there are a smoke detector and a heat detector together with a pump that may be activated in case of fire. We assume that these sensors and the actuator are controlled by a smart device associated to the room, called *Room controller*. For example, each of the k rooms of the Library has its own controller R_{L_i} (with $i \in [1, k]$). Furthermore, at each floor there is a device working as *Floor controller* (e.g. at the ground floor there is the Library controller L) that receives the data sensed in each room, elaborates them to detect possible fires and sends them to the control room.

Also, each Floor controller is equipped with a sensor that checks the water level of the suppression system at that floor.

The Control Room receives data from each floor and, in case of fire, raises the alarm, by calling the Fire Station, and triggers the evacuation plan.

The IoT-LySa *model.* The IoT-LySa model of the network N of our building is in Table 1. It has a finite number of nodes running in parallel (the operator $|$ denotes the parallel composition of nodes). A node represents a smart device as a Room controller, e.g. R_{L_1} for the first room of the Library, a Floor controller, e.g. L for the Library, and the Control Room CR. Furthermore, each node is uniquely identified by a label ℓ and it consists of control processes (the software) and, possibly, of sensors and actuators (the cyber-physical components).

In IoT-LySa communication is multi-party: each node sends information (in the form of tuples of terms) to a set of nodes, provided that they are within the same transmission range. Outputs and inputs must match to allow communication. In more detail, the output $\langle\langle E_1, \cdots, E_k \rangle\rangle \triangleright L$ represents that the tuple $E_1, \cdots, E_k$ is sent to the nodes with labels in L. The input is instead modelled as $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k)$ and embeds pattern matching. In receiving an output tuple $E'_1, \cdots, E'_k$ of the same size (arity), the communication succeeds provided that the first j elements of the output match the corresponding first elements of the input (i.e. $E_1 = E'_1, \cdots, E_j = E'_j$), and then the variables occurring in the input are bound to the corresponding terms in the output. In other words, our primitive for input tests the components of the received message. Suppose e.g. to have a process P waiting a message that P knows to include the value v, together with a datum that is not known from P. The input pattern tuple would be: $(v; x)$. If P receives the matching tuple $\langle v, d \rangle$, the variable x is bound to v, since the first component of the tuple matches with the corresponding value. Finally, note that terms are uniquely annotated to support the Control Flow Analysis (see the next sections).

The Network of Smart Devices

$$
\begin{array}{ll}
N & = CR \mid L \mid O \mid IT \mid \\
 & \quad R_{L1} \mid \ldots \mid R_{Lk} \mid R_{O1} \mid \ldots \mid R_{Ok} \mid R_{I1} \mid \ldots \mid R_{Ik}
\end{array}
$$

Room controllers of the Library with $i \in [1,k]$

$$
\begin{array}{ll}
R_{Li} & = \ell_{Li} : [P_{Li} \parallel S_{Li1} \parallel S_{Li2} \parallel B_{Li}] \\
P_{Li} & = \mu h.(z_{Li1}^{r_{Li1}} := 1_{Li}^{s_{Li1}}).(z_{Li2}^{r_{Li2}} := 2_{Li}^{s_{Li2}}). \\
 & \quad \langle\!\langle i, z_{Li1}^{r_{Li1}}, z_{Li2}^{r_{Li2}} \rangle\!\rangle \triangleright \{\ell_{Library}\}.(z_{check_{Li}}^{r_{ch_{Li}}}).\langle z_{check_{Li}}^{r_{ch_{Li}}}, i_{Li}, Go \rangle.h \\
S_{Li1} & = \mu h.(\tau.1_{Li} := s_{Li}).\tau.h \\
S_{Li2} & = \mu h.(\tau.2_{Li} := h_{Li}).\tau.h \\
A_{Li} & = \mu h.(\!|i_{Li}, \{\mathsf{Go}\}).\tau.h
\end{array}
$$

Floor controller of the Library

$$
\begin{array}{ll}
L & = \ell_L : [P_L \parallel S_L \parallel B_L] \\
P_L & = \mu h.(w_L^{f_L} := 1_L^{s_L}).check_L^{f_{ch_L}} := w_L^{f_L} \geq^{g_L} th_L.\langle check_L^{f_{ch_L}}, i_L, Refill \rangle. \\
 & \quad (1; w_{L11}^{f_{L11}}, w_{L12}^{f_{L12}}). \\
 & \quad check_{L1}^{f_{L1}} := and_{L1}^{a_{L1}}(check_L^{f_{ch_L}}, or_{L1}^{o_{L1}}(w_{L11}^{f_{L11}} \geq^{g_{L11}} th_{L11}, w_{L12}^{f_{L12}} \geq^{g_{L12}} th_{L12})). \\
 & \quad \langle\!\langle check_{L1}^{f_{L1}} \rangle\!\rangle \triangleright \{\ell_{L_1}\}. \\
 & \quad \ldots \\
 & \quad (k; w_{Lk1}^{f_{Lk1}}, w_{Lk2}^{f_{Lk1}}). \\
 & \quad check_{Lk}^{f_{Lk}} := and_{Lk}^{a_{Lk}}(check_L^{f_{ch_L}}, or_{Lk}^{o_{Lk}}(w_{Lk1}^{f_{Lk1}} \geq^{g_{Lk1}} th_{Lk1}, w_{Lk2}^{f_{Lk2}} \geq^{g_{Lk2}} th_{Lk2})). \\
 & \quad \langle\!\langle check_{Lk}^{f_{Lk}} \rangle\!\rangle \triangleright \{\ell_{L_k}\}. \\
 & \quad \langle\!\langle L, p^{f_P}(check_{L1}^{f_{L1}}, ..., check_{Lk}^{f_{Lk}}) \rangle\!\rangle \triangleright \{\ell_{L_{CR}}\}.h \\
S_L & = \mu h.(\tau.1_L := s_L).\tau.h \\
A_L & = \mu h.(\!|i_L, \{\mathsf{Refill}\}).\tau.h
\end{array}
$$

Room controller and Floor controller of the Office area with $i \in [1,k]$

$$
\begin{array}{ll}
R_{Oi} & = \ldots \\
O & = \ldots
\end{array}
$$

Room controller and Floor controller of the IT Center with $i \in [1,k]$

$$
\begin{array}{ll}
R_{Ii} & = \ldots \\
IT & = \ldots
\end{array}
$$

Control Room

$$
\begin{array}{ll}
CR & = \ell_{CR} : [P_{CR} \parallel A_{CR} \parallel B_{CR}] \\
P_{CR} & = \mu h.(L, x_{check_L}^{c_{ch_L}}).alarm_L^{c_{al_L}} := (x_{check_L}^{c_{ch_L}} \geq^{cr_L} th_L).\langle alarm_L{}^{c_{al_L}}, 1_{CR}, ring \rangle \parallel \\
 & \quad (O, x_{check_O}^{c_{ch_O}}).alarm_O^{c_{al_O}} := (x_{check_O}^{c_{ch_O}} \geq^{cr_O} th_O).\langle alarm_O{}^{c_{al_O}}, 1_{CR}, ring \rangle \parallel \\
 & \quad (I, x_{check_I}^{c_{ch_I}}).alarm_I^{c_{al_I}} := (x_{check_I}^{c_{ch_I}} \geq^{cr_I} th_I).\langle alarm_I{}^{c_{al_I}}, 1_{CR}, ring \rangle.h \\
A_{CR} & = \mu h.(1_{CR}, \{\mathsf{Ring}\}).\tau.h
\end{array}
$$

Table 1. Fire Detection System: the controllers of Office area and IT Center are omitted because similar to those of the Library.

We first examine the nodes R_{Li}, located in the rooms of the Library:

$$R_{Li} = \ell_{Li} : [P_{Li} \parallel S_{Li1} \parallel S_{Li2} \parallel B_{Li}]$$

where the label ℓ_{Li} uniquely identifies the node. The control process P_{Li} represents the logic of the device and manages the smoking sensor S_{Li1}, the heating sensor S_{Li1} and the suppression actuator A_{Li}. Whereas B_{Li} represents those node components we are not interested in here, such as the device store Σ_{Li}. All these components run in parallel (this is the meaning of the parallel composition operator $\parallel$ for components inside nodes).

Intuitively, each node R_{Li} collects the data from its sensors, transmits them to the Library controller L, and waits for its decision. In IoT-LySa, sensors communicate the sensed data to the control processes of the node by storing them in their reserved location $1^{s_{L11}}_{L11}$ and $1^{s_{L12}}_{L12}$ of the shared store. The action τ denotes internal actions of the sensor we are not interested in modelling. The construct $\mu h.$ denotes the iterative behaviour of processes, of sensors and actuators, where h is the iteration variable.

More in detail, the control process P_{Li}: (i) stores in the variables $z^{r_{Li1}}_{Li1}$ and $z^{r_{Li2}}_{Li2}$ (where r_{Li1} and r_{Li2} are the variable annotations) the data collected by the smoking and heating sensors, through the two assignments: $z^{r_{Li1}}_{Li1} := 1^{s_{L11}}_{L11}$ and $z^{r_{Li2}}_{Li2} := 1^{s_{L12}}_{L12}$; (ii) transmits the collected data to the Library controller L, with the output $\langle\!\langle i, z^{r_{Li1}}_{Li1}, z^{r_{Li2}}_{Li2} \rangle\!\rangle \triangleright \{\ell_{Library}\}$; (iii) waits for the decision of the Library controller on the input $(z^{r_{ch_{Li}}}_{check_{Li}})$; (iv) forwards the decision to the suppression actuator A_{Li}, with the command $\langle z^{r_{ch_{Li}}}_{check_{Li}}, i_{Li}, Go \rangle$: if $z^{r_{ch_{Li}}}_{check_{Li}}$ is true, the water suppression (action Go) is activated.

In the Library controller L, the process P_L: (i) stores in the variables $w^{f_L}_L$ the data provided by the water level sensor; (ii) checks whether the water level is above a given threshold th_L and if *Refill* is necessary, gives the command $\langle checkL^{f_{ch_L}}, i_L, Refill \rangle$ to the actuator A_L; (iii) waits for the data of each room R_{Li}, with the input $(i; w^{f_{L11}}_{Li1}, w^{f_{Li2}}_{Li2})$; (iv) elaborates the water level and the values received in a variable $check^{f_{Li}}_{Li}$ and provides a decision on the suppression or not in the room R_{Li}; (v) sends to the Control Room Node CR the aggregation p of all the variables $check^{f_{Li}}_{Li}$ previously collected.

Finally, the Control Room node CR: (i) collects the result of aggregation from each floor and puts it in the variable $x^{c_{ch_B}}_{check_B}$ (with $B = L, O, I$); (ii) according to some threshold values, decide to activate an alarm call to the Fire Station.

Again B_L and B_{CR} abstract other components we are not interested in, among which, their stores Σ_L and Σ_{CR}.

Security analysis. Once having specified a system, a designer can use the CFA to statically predict its behaviour. The CFA predicts the "shape" of the data that nodes may exchange and elaborate across the network.

In particular, the CFA can predict, for each piece of data, from which sensors that piece can be derived and which manipulations it may be subjected to. Consider, e.g. the first sensor S_{Li1} of the i-th Library Room controller (the second sensor S_{Li2} is treated analogously). We abstractly represent data produced by

this sensor with the term $1_{Li1}^{s_{Li1}}$, encoding both the identity of the sensor S_{Li1} and of the corresponding node L_i. The term $g_{Li1}(1_{Li1}^{s_{Li1}}, th_{L11})$ denotes instead a piece of data that is obtained by applying the logical function *greater than* (abbreviated with g) to the data from the sensor $1_{Li1}^{s_{Li1}}$ and the constant th_{L11}.

Furthermore, the analysis approximates the trajectories across the network of each piece of data. For instance, we can discover that the data $1_{Lij}^{s_{Lij}}$ collected by the sensors S_{Lij} (with $j \in [1,2]$]) may traverse the path $S_{Lij}, \ell_{Li}, \ell_L, \ell_{Li}$ meaning that they are generated in the sensor S_{Lij}, go to the node ℓ_L and then come back to ℓ_{Li}. Similarly, data produced by the sensor S_L in the node labelled ℓ_L are abstractly represented by the term $1_L^{s_L}$ and may traverse the path S_L, ℓ_L.

Finally, the analysis can also predict on which data the trigger of an actuator may depend on.

By using the CFA results, system designers can investigate the security and robustness of their system through a "what if" reasoning.

For example, in our scenario assume that some nodes can be attacked and that data passing through them tampered with. We would like to study how this may impact on the capacity of the system to correctly trigger the actuators. Consider the suppression system of the generic room controller R_{L_i} of the Library floor. The question we would like to answer is: which are the data that once altered can affect the trigger of the command *Go*? We know that this command is given when the value of the variable $z_{check_{Li}}$ is true (the condition of the trigger), thus, knowing how this value is computed and from which nodes its components come is critical. The analysis predicts (see Section 4 for details) that a possible way to compute the relevant value is the following

$$and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1}))).$$

This "abstract" term encodes the fact that the condition of the trigger depends on the values of the sensors in the node R_{L_i} and L ($1_L^{s_L}$, $1_{Li1}^{s_{Li1}}$ and $1_{Li2}^{s_{Li2}}$), and on a suitable combination of the results of their comparisons with the corresponding thresholds (e.g. g_L, and_{Li}, or_{Li}) carried out by the node L. This information can be easily retrieved by looking at the labels on each sub-term, which indicate the location where that term is introduced and computed. In a sense the "abstract" term encodes the supply chain of the fire detection system.

As mentioned above, for each piece of data the analysis computes its possible trajectories inside the system: the data collected by the sensors S_{Lij} (with $j \in [1,2]$]) may traverse the path $S_{Lij}, \ell_{Li}, \ell_L, \ell_{Li}$ and those produced by the sensor S_L in the node labelled ℓ_L may traverse the path S_L, ℓ_L.

To simplify the reasoning on the analysis results, one can build a graph representation as the one of Fig. 1. The root represents the actuator whose activation we want to reason about; the node under the root is the trigger condition; the other nodes represent the aggregation functions and the data sensors that contribute to the evaluation of the relevant condition. Moreover, the nodes are labelled so as to record in which node of the network the corresponding computations occur. Intuitively, the actuation *Go* depends on the evaluation of the abstract value: $and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1}))),$

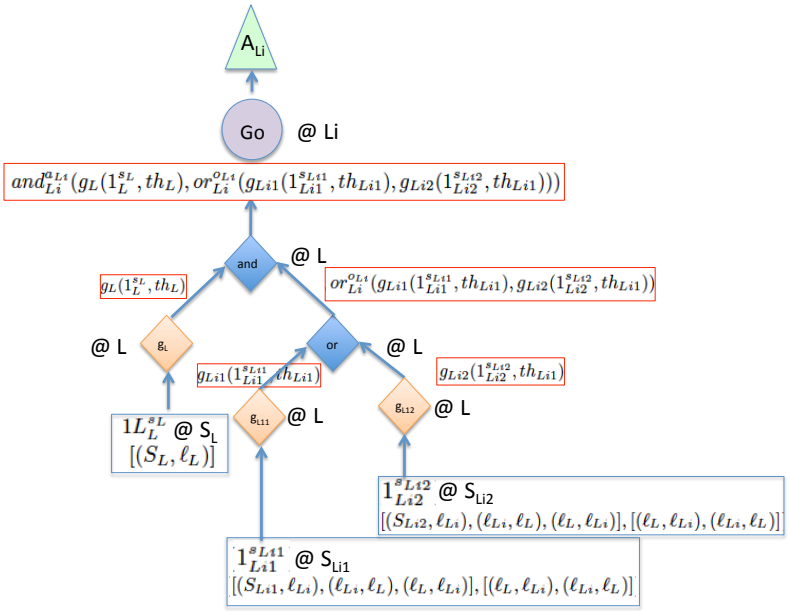

Fig. 1. Composition data graph

that, in turn, depends on the evaluation of the logic conjunction of the results of the two branches: $g_L(1^{s_L}_L, th_L)$ and $or^{o_{Li}}_{Li}(g_{Li1}(1^{s_{Li1}}_{Li1}, th_{Li1}), g_{Li2}(1^{s_{Li2}}_{Li2}, th_{Li1}))$. The first depends on the evaluation of the function g_L applied to the data coming from the sensor S_L, while the second depends from the logic disjunction of the outputs of the two further branches $g_{Li1}(1^{s_{Li1}}_{Li1}, th_{Li1})$ and $g_{Li2}(1^{s_{Li2}}_{Li2}, th_{Li2})$, resulting from the application of the functions g_{Li1} and g_{Li2} to the data coming from the sensors S_{li1} and S_{Li2}, respectively.

Using this graph one can study how an attacker can operate to force a specific behaviour of the actuator, and which nodes are involved. E.g. in case of fire, one can prevent activating of the fire suppression or, on the contrary, force it in order to flood the rooms. Note that although similar in shape to Attack Trees [23], our graphs have a different goal: while the first represents the steps of an attack (the goal being the root node and different ways of achieving it as leaf nodes), we represent in the nodes the components of a system to attack and their dependencies.

For the sake of simplicity, hereafter we focus on the structure induced by the AND and OR functions in the graph, and neglect the comparison functions g_L, g_{Li1}, g_{Li2}. Suppose the attacker wants to impair the trigger of *Go*. This can be done by complementing the boolean value $and^{a_{Li}}_{Li}(g_L(1^{s_L}_L, th_L), or^{o_{Li}}_{Li}(g_{Li1}(1^{s_{Li1}}_{Li1}, th_{Li1}), g_{Li2}(1^{s_{Li2}}_{Li2}, th_{Li1})))$. To turn the value `true` to `false`, it suffices that the attacker forces just one AND branch to `false`. As a consequence, an attacker can:

- tamper the sensors: the sensor S_L in order to provide a `false` on the left branch, or both the sensors S_{Li1} and S_{Li2} in order to provide a false on the `true` branch;
- attack a node: the node L in order to alter the aggregation of data coming from its sensor and/or from the node L_i or the node L_i in order to directly alter the activation.

Of course, the two kinds of attacks require different efforts for the attacker. For instance it could be easier to tamper a sensor in a Library room, where the access is less restricted than in other areas of the building. To estimate these efforts a designer can provide a security score ϕ that measures for each sensor S and for each node N the cost of attacking it. In particular, the designer can use a technique similar to the one in [1] and understand which are the more critical nodes and therefore where security countermeasures are more crucial.

3 Overview of IoT-LySA

We briefly present a version of IoT-LySA [4,3,9], a specification language recently proposed for designing IoT systems. It is, in turn, an adaption of LySA [2], a process calculus introduced to specify and analyse cryptographic protocols and checking their security properties (see e.g. [14,13]).

The calculus IoT-LySA differs from other process algebraic approaches introduced to model IoT systems, e.g. [16,17,18], because it provides a design framework that includes a static semantics to support verification techniques and tools for checking properties of IoT applications.

Syntax Systems in IoT-LySA are pools of nodes (things), each hosting a shared store for internal communication, sensors and actuators, and a finite number of control processes that detail how data are to be processed and exchanged. We assume that each sensor (actuator) in a node with label ℓ is uniquely identified by an index $i \in \mathcal{I}_\ell$ ($j \in \mathcal{J}_\ell$, resp). A sensor is an active entity that reads data from the physical environment at its own fixed rate, and deposits them in the local store of the node. Actuators are instead passive: they just wait for a command to become active and operate on the environment. Data are represented by terms, which carry annotations $a, a', a_i, ... \in \mathcal{A}$ identifying their occurrences. Annotations are used in the analysis and do not affect the dynamic semantics in Table 5. We assume the existence of a finite set $\mathcal{K}$ of secret keys owned by nodes, exchanged at deployment time in a secure way, as it is often the case [24]. The encryption function $\{E_1, \cdots, E_r\}_{k_0}$ returns the result of encrypting values E_i for $i \in [1, r]$ under the shared key k_0. We assume to have perfect cryptography. The term $f(E_1, \cdots, E_r)$ is the application of function f to r arguments; we assume as given a set of primitive functions, typically for aggregating or comparing values. We assume the sets $\mathcal{V}$, $\mathcal{I}_\ell$, $\mathcal{J}_\ell$, $\mathcal{K}$ be pairwise disjoint.

The syntax of systems of nodes and of its components is as follows.

$\mathcal{N} \ni N ::=$ *systems of nodes*		$\mathcal{B} \ni B ::=$ *node components*	
$\mathsf{0}$	empty system	Σ_ℓ	node store
$\ell : [B]$	single node ($\ell \in \mathcal{L}$)	P	process
$N_1 \mid N_2$	par. composition	S	sensor (label $i \in \mathcal{I}_\ell$)
		A	actuator (label $j \in \mathcal{J}_\ell$)
		$B \parallel B$	par. composition

$\mathcal{N} \ni N ::=$	*systems of nodes*	
	$\mathsf{0}$	empty system
	$\ell : [B]$	single node ($\ell \in \mathcal{L}$, the set of labels)
	$N_1 \mid N_2$	parallel composition of nodes
$\mathcal{B} \ni B ::=$	*node components*	
	Σ_ℓ	node store
	P	process
	S	sensor, with a unique identifier $i \in \mathcal{I}_\ell$
	A	actuator, with a unique identifier $j \in \mathcal{J}_\ell$
	$B \parallel B$	parallel composition of node components

Each node $\ell : [B]$ is uniquely identified by a label $\ell \in \mathcal{L}$ that may represent further information on the node (e.g. node location). Sets of nodes are described through the (associative and commutative) operator $\mid$ for parallel composition. The system $\mathsf{0}$ has no nodes. Inside a node $\ell : [B]$ there is a finite set of components combined by means of the parallel operator $\parallel$. We impose that there is a *single* store $\Sigma_\ell : \mathcal{X} \cup \mathcal{I}_\ell \to \mathcal{V}$, where $\mathcal{X}, \mathcal{V}$ are the sets of variables and of values (integers, booleans, ...), respectively.

The store is essentially an array whose indexes are variables and sensors identifiers $i \in \mathcal{I}_\ell$ (no need of α-conversions). We assume that store accesses are atomic, e.g. through CAS instructions [15]. The other node components are control processes P, and sensors S (less than $\#(\mathcal{I}_\ell)$), and actuators A (less than $\#(\mathcal{J}_\ell)$) the actions of which are in Act. The syntax of processes is as follows.

$P ::=$		
	$\mathsf{0}$	inactive process
	$\langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L.\, P$	asynchronous multi-output $L \subseteq \mathcal{L}$
	$(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_r)^X.\, P$	input (with matching and tag)
	$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_r\}_{k_0}\ \mathsf{in}\ P$	decryption with key k_0 (with match.)
	$E ? P : Q$	conditional statement
	h	iteration variable
	$\mu h.\, P$	tail iteration
	$x^a := E.\, P$	assignment to $x \in \mathcal{X}$
	$\langle E, j, \gamma \rangle.\, P$	output of action γ to actuator j on condition E

The prefix $\langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L$ represents a simple form of multi-party communication: the tuple obtained by evaluating $E_1, \ldots, E_r$ is asynchronously sent to the nodes with labels in L that are "compatible" (according to, among other attributes, a proximity-based notion). The input prefix $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_r)$ receives a r-tuple, provided that its first j elements match the corresponding

input ones, and then assigns the variables (after ";") to the received values. Otherwise, the r-tuple is not accepted. A process repeats its behaviour, when defined through the tail iteration construct $\mu h.P$ (where h is the iteration variable). The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_r\}_{k_0}\ \mathsf{in}\ P$ tries to decrypt the result of the expression E with the shared key $k_0 \in \mathcal{K}$. Also in this case, if the pattern matching succeeds, the process continues as P and the variables $x_{j+1}, \ldots, x_r$ are suitably assigned.
Sensors and actuators have the form:

$S ::=$ *sensors*		$A ::=$ *actuators*	
$\mathsf{0}$	inactive sensor	$\mathsf{0}$	inactive actuator
$\tau.S$	internal action	$\tau.A$	internal action
$i := v.\, S$	store of $v \in \mathcal{V}$	$(\!\lvert j, \Gamma \rvert\!).\, A$	command for actuator j ($\Gamma \subseteq Act$)
	by the i^{th} sensor	$\gamma.A$	triggered action ($\gamma \in Act$)
h	iteration var.	h	iteration var.
$\mu\, h\, .\, S$	tail iteration	$\mu\, h\, .\, A$	tail iteration

A sensor can perform an internal action τ or put the value v, gathered from the environment, into its store location i. An actuator can perform an internal action τ or execute one of its actions γ, received from its controlling process. Note that in the present version of IoT-LySA the process triggers the action γ of the actuator, provided the related condition E is satisfied. This construct emphasises that actuation is due to a decision based on the aggregation and elaboration of collected data. Sensors and actuators can iterate. For simplicity, here we neither provide an explicit operation to read data from the environment, nor we describe the impact of actuator actions on the environment (see [9]).

Finally, the syntax of terms is as follows:

$\mathcal{E} \ni E ::=$ *annotated terms*		$\mathcal{M} \ni M ::=$ *terms*	
M^a	with $a \in \mathcal{A}$	v	value ($v \in \mathcal{V}$)
		i	sensor location ($i \in \mathcal{I}_\ell$)
		x	
		$\{E_1, \cdots, E_r\}_{k_0}$	encryption with key $k_0 \in \mathcal{K}$
		$f(E_1, \cdots, E_r)$	function on data

The encryption function $\{E_1, \cdots, E_k\}_{k_0}$ returns the result of encrypting values E_i for $i \in [1, k]$ under k_0 representing a shared key in $\mathcal{K}$. The term $f(E_1, \cdots, E_n)$ is the application of function f to n arguments; we assume given a set of primitive functions, typically for aggregating or comparing values, be them computed or representing data in the environment.

Operational Semantics The semantics is based on a standard structural congruence and a two-level *reduction relation* $\rightarrow$ defined as the least relation on nodes and its components, where we assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms. As examples of semantic rules, we show the rules (Ev-out) and (Multi-com) in Table 2, that drive asynchronous IoT-LySA multi-communications, and the rules (A-com1) and (A-com2) used to communicate

$$\text{(Ev-out)}\quad \frac{\bigwedge_{i=1}^{r} v_i = [\![E_i]\!]_{\Sigma}}{\Sigma \parallel \langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L.\, P \parallel B \;\rightarrow\; \Sigma \parallel \langle\!\langle v_1, \cdots, v_r \rangle\!\rangle \triangleright L.\mathtt{0} \parallel P \parallel B}$$

$$\text{(Multi-com)}\quad \frac{\ell_2 \in L \wedge \;\; Comp(\ell_1, \ell_2) \wedge \;\; \bigwedge_{i=1}^{j} v_i = [\![E_i]\!]_{\Sigma_2}}{\begin{array}{l}\ell_1 : [\langle\!\langle v_1, \cdots, v_r \rangle\!\rangle \triangleright L.\, \mathtt{0} \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2 \parallel (E_1, \cdots, E_j; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}).Q \parallel B_2] \;\rightarrow \\ \ell_1 : [\langle\!\langle v_1, \cdots, v_r \rangle\!\rangle \triangleright L \setminus \{\ell_2\}.\, \mathtt{0} \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2\{v_{j+1}/x_{j+1}, \cdots, v_r/x_r\} \parallel Q \parallel B_2]\end{array}}$$

$$\text{(A-com1)}\quad \frac{\gamma \in \Gamma \wedge [\![E]\!]_{\Sigma} = \mathtt{true}}{\langle E, j, \gamma \rangle.\, P \parallel (\!|j, \Gamma|\!).\, A \parallel B \;\rightarrow\; P \parallel \gamma.\, A \parallel B} \qquad \text{(A-com2)}\quad \frac{\gamma \in \Gamma \wedge [\![E]\!]_{\Sigma} = \mathtt{false}}{\langle E, j, \gamma \rangle.\, P \parallel (\!|j, \Gamma|\!).\, A \parallel B \;\rightarrow\; P \parallel (\!|j, \Gamma|\!).\, A \parallel B}$$

Table 2. Communication semantic rules.

with actuators. The complete semantics is in Appendix, where however we omit the rules handling errors, e.g. when a node fails receiving a message.

In the (Ev-out) rule, to send a message $\langle\!\langle v_1, ..., v_r \rangle\!\rangle$ obtained by evaluating $\langle\!\langle E_1, ..., E_r \rangle\!\rangle$, a node with label ℓ spawns a new process, running in parallel with the continuation P; this new process offers the evaluated tuple to all the receivers with labels in L. In the (Multi-com) rule, the message coming from ℓ_1 is received by a node labelled ℓ_2, provided that: (i) ℓ_2 belongs to the set L of possible receivers, (ii) the two nodes satisfy a compatibility predicate $Comp$ (e.g. when they are in the same transmission range), and (iii) that the first j values match with the evaluations of the first j terms in the input. When this match succeeds the variables after the semicolon ";" are assigned to corresponding values of the received tuple. Moreover, the label ℓ_2 is removed by the set of receivers L of the tuple. The spawned process terminates when all the receivers have received the message ($L = \emptyset$).

In the rules (A-com1) and (A-com2) a process with prefix $\langle E, j, \gamma \rangle$ commands the j^{th} actuator to perform the action γ, if it is one of its actions and it the condition E is true, according to the standard denotational intepretation $[\![E]\!]_{\Sigma}$.

4 Control Flow Analysis

Here we present a variant of the CFA of [7], following the same schema of the ones in [4,8]. It approximates the abstract behaviour of a system of nodes and tracks the trajectories of data. Intuitively, abstract values "symbolically" represent run-time data so as to encode where they have been introduced and elaborated. Here the analysis has a new component to track the conditions that may trigger actuators. Finally, we show how to use the CFA results to check which are the possible trajectories of the data and which of these data affect actuations.

Abstract values Abstract values represent data from sensors and resulting from aggregation functions and encryptions. Furthermore, for analysis reasons we also record into abstract values the annotations of the expression where they are

generated. During the execution nodes may generate encrypted terms with an arbitrarily nesting level, due to recursion. To deal with them, in the analysis we also introduce the special abstract values $(\top, a)$ denoting terms with a depth greater than a threshold d. In the analysis specification we maintain the depth of abstract values smaller than d using the cut function $\lfloor - \rfloor_d$. It is inductively defined on the structure of the abstract value and cut it when the relevant depth is reached. Formally, abstract values are defined as follows, where $a \in \mathcal{A}$.

$\hat{\mathcal{V}} \ni \hat{v} ::=$ *abstract terms*	
$(\top, a)$	value denoting cut
(v, a)	value for clear data
$(f(\hat{v}_1, \cdots, \hat{v}_n), a)$	value for aggregated data
$(\{\hat{v}_1, \cdots, \hat{v}_n\}_{k_0}, a)$	value for encrypted data

Abstract values are pairs where the first component is a value and the second one records the annotation of the expression where the value was computed. In particular, v abstracts the concrete value from sensors or computed by a function in the concrete semantics; $f(\hat{v}_1, \cdots, \hat{v}_n)$ represents a value obtained by applying the aggregation function f to $\hat{v}_1, \cdots, \hat{v}_n$; $\{\hat{v}_1, \cdots, \hat{v}_n\}^a_{k_0}$ abstracts encrypted data.

To simplify the notation, hereafter we write abstract values (v, a) as v^a and indicate with $_{\downarrow_i}$ the projection function on the i^{th} component of the pair. We naturally extend the projection to sets, i.e. $\hat{V}_{\downarrow_i} = \{\hat{v}_{\downarrow_i} | \hat{v} \in \hat{V}\}$, where $\hat{V} \subseteq \hat{\mathcal{V}}$.

We denote with $\mathbf{A}$ the function that recursively extracts all the annotations from an abstract value, e.g. $\mathbf{A}(f(v_1^{a_1}, v_2^{a_2}), a) = \{a_1, a_2, a\}$.

Definition 1. *Give an abstract value $\hat{v} \in \hat{\mathcal{V}}$, we define the set of labels $\mathbf{A}(\hat{v})$ inductively as follows.*

- $\mathbf{A}(\top, a) = \mathbf{A}(v, a) = \{a\}$
- $\mathbf{A}(f(\hat{v}_1, \cdots, \hat{v}_n), a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v}_i)$
- $\mathbf{A}(\{\hat{v}_1, \cdots, \hat{v}_n\}_{k_0}, a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v}_i)$

CFA validation and correctness Here, we define our CFA that approximates system behaviours, e.g. communications and exchanged data, and in particular, micro-trajectories of data. As usual with the CFA, the analysis is specified through a set of inference rules expressing when the analysis results are valid. The analysis result is a tuple $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha)$ (a pair $(\widehat{\Sigma}, \Theta)$ when analysing a term), called *estimate* for N (for E), where $\widehat{\Sigma}$, κ, Θ, T, and α are the following *abstract domains*:

- the disjoint union $\widehat{\Sigma} = \bigcup_{\ell \in \mathcal{L}} \hat{\Sigma}_\ell$ where each function $\hat{\Sigma}_\ell : \mathcal{X} \cup \mathcal{I}_\ell \rightarrow 2^{\widehat{\mathcal{V}}}$ maps a given sensor in $\mathcal{I}_\ell$ or a given variable in $\mathcal{X}$ to a set of abstract values;
- a set $\kappa : \mathcal{L} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ of the messages that may be received by the node ℓ;
- a set $\Theta : \mathcal{L} \rightarrow \mathcal{A} \rightarrow 2^{\widehat{\mathcal{V}}}$ of the information of the actual values computed by each labelled term M^a in a given node ℓ, at run time;

- a set $T = \mathcal{A} \to (\mathcal{L} \times \mathcal{L})$ of possible micro-trajectories related to the abstract values;
- a set $\alpha : \mathcal{L} \to \Gamma \ \to \ 2^{\widehat{\mathcal{V}}}$ that connects the abstract values that can reach the condition related to an action γ.

With respect to the previous analyses of IoT-LySA, such as the ones in [4,7], the component α is new, and also the combined use of the above five components is new and allows us to potentially integrate the present CFA with the previous ones. For simplicity, we do not have here the component that tracks which output tuples that may be accepted in input as in [7]. Note that the component α allows us to determine the trajectories and the provenance of the values reaching the annotated condition. Using such information, we can therefore reason on how critical decisions may depend on them.

An available estimate has to be validated correct. This requires that it satisfies the judgements defined according to the syntax of nodes, node components and terms, fully presented in Appendix (see Tables 6 and 7). They are defined by a set of clauses. Here, we just show some examples.

The judgements for labelled terms have the form $(\widehat{\Sigma}, \Theta) \models_\ell M^a$. Intuitively, they require $\Theta(\ell)(a)$ includes all the abstract values $\hat{v}$ associated to M^a, e.g. if the term is a sensor identifier i^a, $\Theta(\ell)(a)$ includes (i, a) and the micro-trajectory (S_i, ℓ) belongs to $T(a)$, if the term is a variable x^a, $\Theta(\ell)(a)$ includes the abstract values bound to x collected in $\widehat{\Sigma}_\ell$. The judgements for nodes have the form $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models N$. The rule for a single node $\ell : [B]$ requires that B is analysed with judgements $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell B$. As examples of clauses, we consider the clauses for communication in Table 3. An estimate is valid for *multi-output*,

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} \ (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \ \wedge \ (\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell P \ \wedge \\ \forall \hat{v}_1, \cdots, \hat{v}_r : \ \bigwedge_{i=1}^{r} \ \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \forall \ell' \in L : (\ell, \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell')\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell \langle\!\langle M_1^{a_1}, \cdots, M_r^{a_r} \rangle\!\rangle \triangleright L.\, P}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{j} \ (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \ \wedge \\ \forall (\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell) : \ \mathit{Comp}(\ell', \ell) \Rightarrow \\ (\bigwedge_{i=j+1}^{r} \ \hat{v}_i \in \hat{\Sigma}_\ell(x_i) \ \wedge \\ (\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \rho(X) \ \wedge \ \forall a \in \mathbf{A}(\hat{v}_i).(\ell, \ell') \in T(a) \\ \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell P)\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell (M_1^{a_1}, \cdots, M_j^{a_j}; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}).\, P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma}, \Theta) \models_\ell M^a \wedge \\ \Theta(\ell)(a) \subseteq \alpha(\ell)(\gamma) \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell P\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_\ell \langle M^a, j, \gamma \rangle.\, P}$$

Table 3. Communication CFA rules.

if it is valid for the continuation of P and the set of messages communicated

by the node ℓ to each node ℓ' in L, includes all the messages obtained by the evaluation of the r-tuple $\langle\!\langle M_1^{a_1}, \cdots, M_r^{a_r} \rangle\!\rangle$. More precisely, the rule (i) finds the sets $\Theta(\ell)(a_i)$ for each term $M_i^{a_i}$, and (ii) for all tuples of values $(\hat{v}_1, \cdots, \hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$ it checks whether they belong to $\kappa(\ell')$ for each $\ell' \in L$. Symmetrically, the rule for *input* requires that the values inside messages that can be sent to the node ℓ, passing the pattern matching, are included in the estimates of the variables $x_{j+1}, \cdots, x_r$. More in detail, the rule analyses each term $M_i^{a_i}$, and requires that for any message that the node with label ℓ can receive, i.e. $(\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_j, \hat{v}_{j+1}, \ldots, \hat{v}_r \rangle\!\rangle)$ in $\kappa(\ell)$, provided that the two nodes can communicate (i.e. $Comp(\ell', \ell)$), the abstract values $\hat{v}_{j+1}, \ldots, \hat{v}_r$ are included in the estimates of $x_{j+1}, \cdots, x_r$. Furthermore, the micro-trajectory (ℓ, ℓ') is recorded in the T component for each annotation related (via $\mathbf{A}$) to the abstract value $\hat{v}_i$, to record that the abstract value $\hat{v}_i$ coming from the node ℓ can reach the node labelled ℓ', e.g. if $\hat{v}_i = (f((v_{i1}, a_{i1}), (v_{i2}, a_{i2})), a_i)$, then the micro-trajectory is recorded in $T(a_i)$, $T(a_{i1})$ and $T(a_{i2})$.

The rule for actuator trigger predicts in the component α that a process at node ℓ may trigger the action γ for the actuator j, based on the abstract values included in the analysis of the term M^a.

Example 1. To better understand how our analysis works, we apply it to the following simple system of three nodes $N_1 \mid N_2 \mid N_3$, where S_1 is a sensor of the first node and P_i' and B_i (with $i = 1, 2, 3$) abstract other components we are not interested in.

$$
\begin{array}{l}
N_1 = \ell_1 : [P_1 \parallel S_1 \parallel B_1] \mid \\
N_2 = \ell_2 : [P_2 \parallel B_2] \mid \\
N_3 = \ell_3 : [P_3 \parallel B_3] \\
P_1 = x^{a1} = 1^{s1}.\langle\!\langle x^{a_1} \rangle\!\rangle \triangleright \ell_2.\, P_1' \\
P_2 = (; x_2^{b_2}).\langle\!\langle f^m(x_2^{b_x}) \rangle\!\rangle \triangleright \ell_3.P_2' \\
P_3 = (; y_3^{c_3}).P_3'
\end{array}
$$

Every valid estimate $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha)$ must include at least the following entries, with $d = 4$.

$$
\begin{array}{lll}
\widehat{\Sigma}_{\ell_1}(x^{a_1}) \supseteq \{1^{s_1}\} & \widehat{\Sigma}_{\ell_2}(x^{b_2}) \supseteq \{1^{s_1}\} & \widehat{\Sigma}_{\ell_3}(y^{c_3}) \supseteq \{f^m(1^{s_1})\} \\
T(s_1) \supseteq \{(S_1, \ell_1)\} & T(a_1) \supseteq \{(\ell_1, \ell_2)\} & T(s_1) \supseteq \{(\ell_2, \ell_3)\} \\
\Theta(\ell_1)(s_1) \supseteq \{1^{s_1}\} & \Theta(\ell_2)(b_2) \supseteq \{f^m(1^{s_1})\} & T(m) \supseteq \{(\ell_2, \ell_3)\} \\
\kappa(\ell_2) \supseteq \{(\ell_1, \langle\!\langle 1^{s_1} \rangle\!\rangle)\} & \kappa(\ell_3) \supseteq \{(\ell_2, \langle\!\langle f^m(1^{s_1}) \rangle\!\rangle)\} & \\
T(s_1) \supseteq \{(\ell_1, \ell_2)\} & &
\end{array}
$$

Indeed, an estimate must satisfy the checks of the CFA rules. The validation of the system requires the validation of each node, i.e. $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models N_i$ and of the processes there included, i.e. $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_{\ell_i} P_i$, with $i = 1, 2, 3$. In particular, the validation of the process P_1, i.e. $x^{a_1} := 1^{s_1}.P_1'$ holds because $1^{s_1} \in \Theta(\ell_1)(s_1)$ and $(S_1, \ell_1) \in T(s_1)$, $1^{s_1} \in \widehat{\Sigma}_{\ell_1}(x^{a_1})$ and the continuation holds as well. In particular, $\langle\!\langle x^{a_1} \rangle\!\rangle \triangleright \{\ell_2\}$ holds because the checks required by CFA clause for output succeed. We can indeed verify that $(\widehat{\Sigma}, \Theta) \models_\ell x^{a_1}$ holds because $1^{s_1} \in \widehat{\Sigma}_{\ell_1}(x^{a_1})$, according to the CFA clause for variables. Furthermore $(\ell_1, \langle\!\langle 1^{s_1} \rangle\!\rangle) \in$

$\kappa(\ell_2)$. This suffices to validate the output, by assuming that the continuation P_1' is validated as well. We have the following instantiation of the clause for output.

$$\frac{1^{s_1} \in \Theta(\ell_1)(s_1) \ \wedge \ (S_1, \ell_1) \in T(s_1)}{(\hat{\Sigma}, \Theta) \models_{\ell_1} 1^{s_1}} \ \wedge \ (\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_{\ell_1} P_1' \ \wedge$$

$$\frac{1^{s_1} \in \Theta(\ell_1)(s_1) \ \Rightarrow (\ell_1, \langle\!\langle 1^{s_1} \rangle\!\rangle) \in \kappa(\ell_2)}{(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_{\ell_1} \langle\!\langle 1^{s_1} \rangle\!\rangle \triangleright \{\ell_2\}. P_1'}$$

Instead $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models_{\ell_1} (; x_2^{b_2}).\langle\!\langle f(x_2^{b_x})^m \rangle\!\rangle \triangleright \ell_3.P_2'$ holds because the checks for the CFA clause for input succeed. From $(\ell_1, \langle\!\langle 1^{s_1} \rangle\!\rangle) \in \kappa(\ell_2)$, we can indeed obtain that $\widehat{\Sigma}_{\ell_2}(x^{b_2}) \supseteq \{1^{s_1}\}$, and that $T(s_1) \supseteq \{(\ell_1, \ell_2)\}$. The other entries can be similarly validated as well.

The following theorem establishes the correctness of our CFA w.r.t. the dynamic semantics. The statement relies on the agreement relation $\bowtie$ between the concrete and the abstract stores. Its definition is immediate, since the analysis only considers the second component of the extended store, i.e. the abstract one: $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$ iff $w \in \mathcal{X} \cup \mathcal{I}_\ell$ such that $\Sigma_\ell^i(w) \neq \perp$ implies $(\Sigma_\ell^i(w))_{\downarrow_2} \in \widehat{\Sigma}_\ell(w)$.

It is also possible to prove the existence of a (minimal) estimate, as in [4], since estimates form a Moore family with a minimal element.

Theorem 1 (Subject reduction). *If $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models N$ and $N \rightarrow N'$ and $\forall \Sigma_\ell^i$ in N it is $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$, then $(\widehat{\Sigma}, \kappa, \Theta, T, \alpha) \models N'$ and $\forall \Sigma_\ell^{i\prime}$ in N' it is $\Sigma_\ell^{i\prime} \bowtie \widehat{\Sigma}_\ell$.*

The proofs follow the same schema of [4]. In particular, we use an instrumented denotational semantics for expressions, the values of which are pairs $\langle v, \hat{v} \rangle$, where v is a concrete value and $\hat{v}$ is the corresponding abstract value. The store (Σ_ℓ^i with an undefined $\perp$ value) is accordingly extended. Our semantics (see Table 5 in Appendix) just uses the projection on the first component.

Checking trajectories We now recall the notion of trajectories of data, in turn composed by micro-trajectories representing a single hop in the communication. It is a slight simplification of the notion presented in [7].

Definition 2. *Given a set of labels $\mathcal{L}$, we define a* micro-trajectory *μ as a pair $(\ell, \ell') \in (\mathcal{L} \times \mathcal{L})$. A* trajectory *$\tau$ is a list of micro-trajectories $[\mu_1, ..., \mu_n]$, such that $\forall \mu_i, \mu_{i+1}$ with $\mu_i = (\ell_i, \ell_i')$ and $\mu_{i+1} = (\ell_{i+1}, \ell_{i+1}')$, $\ell_i' = \ell_{i+1}$.*

A k-trajectory *is a list of micro-trajectories $[\mu_1, ..., \mu_k]$ of length k.*

In our analysis, one starts from a set of micro-trajectories and suitably compose them to obtain longer trajectories, in turn composed. As expected, for composing trajectories the head of the second must be equal to tail of the first. Technically, we use a closure of a set of micro-trajectories, the inductive definition of which follows.

Definition 3. *Given a set of micro-trajectories $S \in (\mathcal{L} \times \mathcal{L})$, its closure $Close(S)$ is defined as*

– $\forall(\ell,\ell') \in S.\ [(\ell,\ell')] \in \mathit{Close}(S)$;
– $\forall[L,(\ell,\ell')],\ [(\ell',\ell''),L''] \in S.\ [L,(\ell,\ell'),(\ell',\ell''),L''] \in \mathit{Close}(S)$.

The set $\mathit{Close}(S)$ contains trajectories of any size. To obtain only the set of k-trajectories it suffices to the subset consisting only of those of length k.

Given a term E annotated by a, the over-approximation of its possible trajectories is obtained by computing the trajectory closure of the set composed by all the micro-trajectories (ℓ,ℓ') in $T(a)$.

$$\mathit{Trajectories}(E^a) = \mathit{Close}(T(a))$$

Therefore, our analysis enables traceability of data. For every exchanged message $\langle\!\langle v_1,\ldots,v_r\rangle\!\rangle$, the CFA keeps track of the possible composition of each of its components and of the paths of each of its components v_i and, in turn, for each v_i it keeps recursively track of the composition and of the paths of the possible data used to compose it. As a corollary of Theorem 1, it follows that κ predicts all the possible inter-node communications, and that our analysis records the micro-trajectory in the T component of each abstract value possibly involved in the communication.

Example 2. Back to our previous example, note that from $T(s_1) \supseteq \{(\ell_1,\ell_2)\}$ and $T(s_1) \supseteq \{(\ell_2,\ell_3)\}$, we can obtain the trajectory $[(\ell_1,\ell_2),(\ell_2,\ell_3)]$, by applying *Close* to $T(s_1)$.

Example 3. Consider now our running example on the fire system network in Section 2. Every valid estimate $(\widehat{\Sigma},\kappa,\Theta,T,\alpha)$ must include at least the entries in Table 4, assuming $d = 4$, and where we overload the symbols g_L, g_{Li1}, g_{Li2}, by meaning both the labels and the comparison functions to check whether the argument are above the given thresholds (they are constants and their labels are omitted for simplicity).
Since we are interested in understanding which data may affect the action *go*, consider $\alpha_{Li}(\mathit{Go})$ that includes, according to the CFA results, the following abstract value

$$and_{Li}^{a_{Li}}(g_L(1_L^{s_L},th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}},th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}},th_{Li1}))).$$

Furthermore, for each sub-term of the abstract value, by using the component T of the analysis, we can retrieve the possible trajectories, in particular for the sensor values with $j = 1,2$ (the subscribed numbers recall their length):

$$\mathit{Trajectories}_3(1_{Lij}^{s_{Lij}}) = \mathit{Close}(T(s_{Lij})) \supseteq \{[(S_{Lij},\ell_{Li}),(\ell_{Li},\ell_L),(\ell_L,\ell_{Li})]\}$$
$$\mathit{Trajectories}_1(s_L) = \mathit{Close}(T(s_L)) \supseteq \{[(S_L,\ell_L)]\}\}$$

With an approach and a technique similar to those in [1], we aim at identifying the possible minimal sets of nodes that must be tampered in order to alter the result of the value the actuation of *Go* depends on and at exploiting the score metric to compare them in order to determine the ones with minimal cost for the attacker.

$\widehat{\Sigma}_{\ell_{Li}}(z_{Lij}^{r_{Lij}}) \supseteq \{1_{Lij}^{s_{Lij}}\},\ \Theta(\ell_{Li})(r_{Lij}) \supseteq \{1_{Li1}^{s_{Lij}}\},\ T(s_{Lij}) \supseteq \{(S_{Lij}, \ell_{Li})\},$
$\kappa(\ell_{L_i}) \supseteq \{(\ell_L, \langle\!\langle and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\rangle\!\rangle)\}$
$\alpha_{Li}(Go) \supseteq \Theta(\ell_{Li})(r_{ch_{Li}}) \supseteq and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\}$
$\widehat{\Sigma}_{\ell_{Li}}(z_{check_{Li}}^{r_{ch_{Li}}}) \supseteq \{and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\}$
$\Theta(\ell_{Li})(r_{ch_{Li}}) \supseteq \{and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\}$
$T(s_{Lij}) \supseteq \{(\ell_L, \ell_{Li})\},$
$\widehat{\Sigma}_{\ell_L}(w_L^{f_L}) \supseteq \{1_L^{s_L}\},\ \Theta(\ell_L)(f_L) \supseteq \{1_L^{s_L}\},\ T(s_L) \supseteq \{(S_L, \ell_L)\}$
$\widehat{\Sigma}_L(check_L^{f_{ch_L}}) \supseteq \{g_L(1_L^{s_L}, th_L)\}, \Theta(\ell_L)(f_{ch_L}) \supseteq \{g_L(1_L^{s_L}, th_L)\}$
$\alpha_L(Refill) \supseteq \Theta(\ell_L)(f_{ch_L}) \supseteq \{g_L(1_L^{s_L}, th_L)\}$
$\widehat{\Sigma}_{\ell_L}(w_{Lij}^{f_{Lij}}) \supseteq \{1_{Lij}^{s_{Lij}}\},\ \Theta(\ell_L)(f_{Lij}) \supseteq \{1_{Lij}^{s_{Lij}}\},\ T(s_{Lij}) \supseteq \{(\ell_{Li}, \ell_L)\},$
$\widehat{\Sigma}_L(check_{Li}^{f_{ch_{Li}}}) \supseteq \{and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\},$
$\Theta(\ell_L)(f_{ch_{Li}}) \supseteq \{and_{Li}^{a_{Li}}(g_L(1_L^{s_L}, th_L), or_{Li}^{o_{Li}}(g_{Li1}(1_{Li1}^{s_{Li1}}, th_{Li1}), g_{Li2}(1_{Li2}^{s_{Li2}}, th_{Li1})))\}$
$\kappa(\ell_L) \supseteq \{(\ell_{L_i}, \langle\!\langle i, 1_{Li1}^{s_{Li1}}, 1_{Li2}^{s_{Li2}}\rangle\!\rangle)\}$
$Trajectories_3(1_{Lij}^{s_{Lij}}) = Close(T(s_{Lij})) \supseteq \{[(S_{Lij}, \ell_{Li}), (\ell_{Li}, \ell_L), (\ell_L, \ell_{Li})]\}$
$Trajectories_1(s_L) = Close(T(s_L)) \supseteq \{[(S_L, \ell_L)]\}\}$
$T(f_p) \supseteq \{[(\ell_L, \ell_{CR})]\}$
$\kappa(\ell_{CR}) \supseteq \{(\ell_L, \langle\!\langle p^{f_p}(and_{L1}^{a_{L1}}(g_L(1_L^{s_L}, th_L), or_{L1}^{o_{L1}}(g_{L11}(1_{L11}^{s_{L11}}, th_{L11}), g_{L12}(1_{L12}^{s_{L12}}, th_{L12}))), ...,$
$and_{Lk}^{a_{Lk}}(g_L(1_L^{s_L}, th_L), or_{Lk}^{o_{Lk}}(g_{Lk1}(1_{Lk1}^{s_{Lk1}}, th_{Lk1}), g_{Lk2}(1_{Lk2}^{s_{Lk2}}, th_{Lk2})))))\rangle\!\rangle)\}$

Table 4. Fire Detection System Analysis: some entries, where $j = 1, 2$.

In the case the attacker would like to tamper the sensors in order to force the result to be `false`, as mentioned above, there are two possible minimal choices, i.e. tampering:

- the sensor S_L in order to alter the sensed data and force $g_L(1_L^{s_L}, th_L)$ to provide a `false` on the left branch, or
- both the sensors S_{Li1} and S_{Li2} in order to provide a `false` on the right branch.

To estimate the cost of the different strategies of attack, we assume that a table of scores is known that associates a score $\phi(S)$, $\phi(\ell)$ to each sensor S and to each node with label ℓ, respectively. Suppose that the scores are:

$$\phi(S_L) = 3 \qquad \phi(S_{Li1}) = 1 \qquad \phi(S_{Li2}) = 1.5$$

Under these hypotheses, for the attacker is more convenient to attack the two sensors in the Room Controller than attacking the sensor of the Library controller. The overall cost of the first option is indeed 2.5, while the cost of the second one is 3.

From this kind of reasoning designers can guide their choices on how to reduce the risk of impairing some critical actuations, e.g. by introducing some redundancy in the sources of data, such as adding new hardware and software components.

5 Conclusions

We started from the modelling language IoT-LySa and from the results of its CFA, and showed how administrators can exploit a graph, built from the analysis results, that represents the data dependencies and the data trajectories. In particular, the graph encodes how the condition that drives a critical actuation depends on the raw data collected by sensors and on the logical and aggregation functions applied to the sensor data. This information allows designers to reason, along the lines of [1], on which data can be altered (by tampering the corresponding nodes) to impact on the capacity of the system to correctly trigger the actuators. Since each node is associated with a score that measures its compromise cost, it is also possible to compare the different solutions and establish the cheapest. Other metrics such as the ones suggested still in [1] can be exploited as well. Then, we discussed how this graph could be used as input to compute different security metrics, e.g. those of [1], for estimating the cost of attacks and devise suitable countermeasures.

Actually, the analysis is quite general and its results can be exploited as a starting point for many other different investigations on the behaviour of a given system. For example, the graph built from the analysis results can be used: to check whether a system respects policies that rule information flows among nodes, by allowing some flows and forbidding others; to carry out a taint analysis for detecting whether critical decisions may depend on tainted data. To this aim, we could integrate our present analysis with the one in [3] in the first case and with the taint analysis of [8] in the second one. Answering to these questions can help designers to detect the potential vulnerabilities related to the presence of dangerous nodes, and can determine possible solutions and mitigation strategies.

A similar Control Flow Analysis is presented in [10]: it is there used to over-approximate the behaviour of KLAIM processes and to track how tuple data can move in the network.

As a future work we plan to fully formalise how to derive the graph from the analysis results. We would like also to compare our approach with that of [19], looking for possible synergies. Its authors also start from a hybrid process calculus for modelling both CPSs and their potential attacks, and propose a threat model that can be used to assess attack tolerance and to estimate the impact of a successful attack.

Another line of future work is linking our approach to that used in [21,22], for ensuring a certain level of quality service of a system even when in the presence of not completely reliable data. In the cited paper, the authors introduce the Quality Calculus that allows defining and reasoning on software components that have a sort of backup plan in case the ideal behaviour fails due to unreliable communication or data.

Finally, since in many IoT system the behaviour of node depends on the computational context they are immersed in, we plan to extend IoT-LySa with constructs for representing contexts along the lines of [11,12], and to study their security along the lines of [5,6].

References

1. Barrère, M., Hankin, C., Nicolaou, N., Eliades, D.G., Parisini, T.: Identifying security-critical cyber-physical components in industrial control systems. CoRR **abs/1905.04796** (2019), `http://arxiv.org/abs/1905.04796`
2. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. Journal of Computer Security **13**(3), 347–390 (2005)
3. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: A step towards checking security in IoT. In: Procs. of ICE 2016. EPTCS, vol. 223, pp. 128–142 (2016)
4. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Where do your IoT ingredients come from? In: Procs. of Coordination 2016. LNCS, vol. 9686, pp. 35–50. Springer (2016)
5. Bodei, C., Degano, P., Galletta, L., Salvatori, F.: Linguistic Mechanisms for Context-aware Security. In: Ciobanu, G., Méry, D. (eds.) ICTAC 2014. LNCS, vol. 8687. Springer (2014)
6. Bodei, C., Degano, P., Galletta, L., Salvatori, F.: Context-aware security: Linguistic mechanisms and static analysis. Journal of Computer Security **24 (4)**, 427–477 (2016)
7. Bodei, C., Galletta, L.: Tracking data trajectories in IoT. In: Mori, P., Furnell, S., Camp, O. (eds.) Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP2019)
8. Bodei, C., Galletta, L.: Tracking sensitive and untrustworthy data in IoT. In: Procs.of the First Italian Conference on Cybersecurity (ITASEC 2017). pp. 38–52. CEUR Vol-1816 (2017)
9. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Tracing where IoT data are collected and aggregated. Logical Methods in Computer Science **13**(3) (2017)
10. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Revealing the trajectories of KLAIM tuples, statically. In: Models, Languages, and Tools for Concurrent and Distributed Programming. Lecture Notes in Computer Science, vol. 11665. Springer (2019)
11. Degano, P., Ferrari, G.L., Galletta, L.: A two-component language for COP. In: Proceedings of 6th International Workshop on Context-Oriented Programming, COP@ECOOP 2014. pp. 6:1–6:7. ACM (2014)
12. Degano, P., Ferrari, G.L., Galletta, L.: A two-component language for adaptation: Design, semantics, and program analysis. IEEE Trans. Software Eng. **42**(6), 505–529 (2016)
13. Gao, H., Bodei, C., Degano, P.: A formal analysis of complex type flaw attacks on security protocols. In: Proc. of AMAST'08. pp. 167–183. LNCS 5140, Springer (2008)
14. Gao, H., Bodei, C., Degano, P., Nielson, H.: A formal analysis for capturing replay attacks in cryptographic protocols. In: Proc. of ASIAN'07. pp. 150–165. LNCS 4846, Springer (2007)
15. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. **13**(1) (1991)
16. Lanese, I., Bedogni, L., Felice, M.D.: Internet of Things: a process calculus approach. In: Procs of the 28th Annual ACM Symposium on Applied Computing, SAC '13. pp. 1339–1346. ACM (2013)
17. Lanotte, R., Merro, M.: A semantic theory of the Internet of Things. In: Procs. of Coordination 2016. LNCS, vol. 9686, pp. 157–174. Springer (2016)

18. Lanotte, R., Merro, M.: A semantic theory of the Internet of Things. Inf. Comput. **259**(1), 72–101 (2018)
19. Lanotte, R., Merro, M., Muradore, R., Viganò, L.: A formal approach to cyber-physical attacks. In: 30th IEEE Computer Security Foundations Symposium. pp. 436–450. IEEE Computer Society (2017)
20. Nicolaou, N., Eliades, D.G., Panayiotou, C.G., Polycarpou, M.M.: Reducing vulnerability to cyber-physical attacks in water distribution networks. In: 2018 International Workshop on Cyber-physical Systems for Smart Water Networks, CySWater@CPSWeek. pp. 16–19. IEEE Computer Society (2018)
21. Nielson, H.R., Nielson, F., Vigo, R.: A calculus for quality. In: Formal Aspects of Component Software, 9th International Symposium, FACS 2012. Lecture Notes in Computer Science, vol. 7684, pp. 188–204. Springer (2013)
22. Nielson, H.R., Nielson, F., Vigo, R.: A calculus of quality for robustness against unreliable communication. J. Log. Algebr. Meth. Program. **84**(5), 611–639 (2015)
23. Schneier, B.: Attack trees. Dr Dobb's Journal **24, 12**, 436–450 (1999)
24. Zillner, T.: ZigBee Exploited (2015), `https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf`

Appendix

Operational Semantics of IoT-LySA

Our reduction semantics is based on the following *Structural congruence* $\equiv$ on nodes and node components. It is standard except for rule (4) that equates a multi-output with no receivers and the inactive process, and for the fact that inactive components of a node are all coalesced.

$$\begin{array}{l}(1)\ (\mathcal{N}/_{\equiv}, |, \mathsf{0}) \text{ is a commutative monoid}\\ (2)\ (\mathcal{B}/_{\equiv}, \|, \mathsf{0}) \text{ is a commutative monoid}\\ (3)\ \mu\, h\,.\, X \equiv X\{\mu\, h\,.\, X/h\} \quad \text{for } X \in \{P, A, S\}\\ (4)\ \langle\langle E_1, \cdots, E_r\rangle\rangle : \emptyset.\ \mathsf{0} \equiv \mathsf{0}\end{array}$$

The two-level *reduction relation* $\rightarrow$ is defined as the least relation on nodes and its components satisfying the set of inference rules in Tables 2 and 5. For the sake of simplicity, we use one relation. We assume the standard denotational interpretation $[\![E]\!]_{\Sigma}$ for evaluating terms.

The first two semantic rules implement the (atomic) asynchronous update of shared variables inside nodes, by using the standard notation $\Sigma\{-/-\}$. According to (S-store), the i^{th} sensor uploads the value v, gathered from the environment, into its store location i. According to (Asgm), a control process updates the variable x with the value of E. The rules for conditional (Cond1) and (Cond2) are as expected. The rule (Act) says that the actuator performs the action γ. Similarly, for the rules (Int) for internal actions for representing activities we are not interested in. The communication rules (Ev-out), (Multi-com), (A-com1) and (A-com2) that drive asynchronous multi-communications and communication with actuators are discussed in Section 3. The rule (Decr) tries to decrypt the result $\{v_1, \cdots, v_r\}_k$ of the evaluation of E with the key k_0, and matches it

$$\text{(S-store)}\quad \frac{}{\Sigma \parallel i^a := v^{a'}.\, S_i \parallel B \to \Sigma\{v/i\} \parallel S_i \parallel B} \qquad \text{(Asgm)}\quad \frac{[\![E]\!]_\Sigma = v}{\Sigma \parallel x^a := E.\, P \parallel B \to \Sigma\{v/x\} \parallel P \parallel B}$$

$$\text{(Cond1)}\quad \frac{[\![E]\!]_\Sigma = \texttt{true}}{\Sigma \parallel E?\, P_1 : P_2 \parallel B \;\to\; \Sigma \parallel P_1 \parallel B} \qquad \text{(Cond2)}\quad \frac{[\![E]\!]_\Sigma = \texttt{false}}{\Sigma \parallel E?\, P_1 : P_2 \parallel B \;\to\; \Sigma \parallel P_2 \parallel B}$$

$$\text{(Act)}\quad \frac{}{\gamma.A \;\to\; A} \qquad \text{(Int)}\quad \frac{}{\tau.\, X \;\to\; X}$$

$$\text{(Decr)}\quad \frac{[\![E]\!]_\Sigma = \{v_1, \cdots, v_r\}_{k_0} \;\wedge\; \bigwedge_{i=1}^{j} v_i = [\![E'_i]\!]_\Sigma}{\Sigma \parallel \mathsf{decrypt}\ E\ \mathsf{as}\ \{E'_1, \cdots, E'_j;\, x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}\}_{k_0}\ \mathsf{in}\ P \parallel B \to \Sigma\{v_{j+1}/x_{j+1}, \cdots, v_r/x_r\} \parallel P \parallel B}$$

$$\text{(Ev-out)}\quad \frac{\bigwedge_{i=1}^{r} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L.\, P \parallel B \;\to\; \Sigma \parallel \langle\!\langle v_1, \cdots, v_r \rangle\!\rangle \triangleright L.0 \parallel P \parallel B}$$

$$\text{(Node)}\quad \frac{B \;\to\; B'}{\ell : [B] \;\to\; \ell : [B']} \qquad \text{(ParN)}\quad \frac{N_1 \to N'_1}{N_1 | N_2 \to N'_1 | N_2} \qquad \text{(ParB)}\quad \frac{B_1 \to B'_1}{B_1 \| B_2 \to B'_1 \| B_2} \qquad \text{(CongrY)}\quad \frac{Y'_1 \equiv Y_1 \to Y_2 \equiv Y'_2}{Y'_1 \to Y'_2}$$

Table 5. Reduction semantics (the upper part on node components, the lower one on nodes), where $X \in \{S, A\}$ and $Y \in \{N, B\}$, without the rules (Ev-out), (Multi-com), (A-com1) and (A-com2), discussed in Table 2.

against the pattern $\{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_r\}_{k_0}$. As for communication, when this match succeeds the variables after the semicolon ";" are assigned to values resulting from the decryption. The last rules propagate reductions across parallel composition ((ParN) and (ParB)) and nodes (Node), while (CongrY) is the standard reduction rule for congruence for nodes and node components.

Control Flow Analysis of IoT-LySA

Our CFA is specified in a logical form through a set of inference rules expressing the validity of the analysis results, where the function $\lfloor - \rfloor_d$ to cut all the terms with a depth greater than a given threshold d, with the special abstract values

$$\frac{(i,a) \in \Theta(\ell)(a) \ \wedge \ (S_i,\ell) \in T(a)}{(\widehat{\Sigma},\Theta) \models_\ell i^a} \qquad \frac{(v,a) \in \Theta(\ell)(a)}{(\widehat{\Sigma},\Theta) \models_\ell v^a} \qquad \frac{\hat{\Sigma}_\ell(x) \subseteq \Theta(\ell)(a)}{(\widehat{\Sigma},\Theta) \models_\ell x^a}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\widehat{\Sigma},\Theta) \models_\ell M_i^{a_i} \wedge \\ \forall\, \hat{v}_1,..,\hat{v}_r : \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \ (\lfloor \{\hat{v}_1,..,\hat{v}_r\}_{k_0} \rfloor_d, a) \in \Theta(\ell)(a)\end{array}}{(\widehat{\Sigma},\Theta) \models_\ell \{M_1^{a_1},..,M_r^{a_r}\}^a_{k_0}}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\widehat{\Sigma},\Theta) \models_\ell M_i \wedge \\ \forall\, \hat{v}_1,..,\hat{v}_r : \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \ (f(\hat{v}_1,..,\hat{v}_r), a) \in \Theta(\ell)(a)\end{array}}{(\widehat{\Sigma},\Theta) \models_\ell f(M_1^{a_1},..,M_r^{a_r})^a}$$

Table 6. Analysis of labelled terms $(\widehat{\Sigma},\Theta) \models_\ell M^a$.

$\top^b$, is defined as follows.

$$\begin{array}{l}\lfloor \top^b \rfloor_d = \top^b \\ \lfloor v^b \rfloor_d = v^b \\ \lfloor \{\hat{v}_1,\cdots,\hat{v}_r\}^b_{k_0} \rfloor_0 = \top^b \\ \lfloor \{\hat{v}_1,\cdots,\hat{v}_r\}^b_{k_0} \rfloor_d = \{\lfloor \hat{v}_1 \rfloor_{d-1},\cdots,\lfloor \hat{v}_r \rfloor_{d-1}\}^b_{k_0} \\ \lfloor f(\hat{v}_1,\cdots,\hat{v}_r) \rfloor_d = f(\lfloor \hat{v}_1 \rfloor_{d-1},\cdots,\lfloor \hat{v}_r \rfloor_{d-1})^b\end{array}$$

The result or *estimate* of our CFA is a tuple $(\widehat{\Sigma},\kappa,\Theta,T,\alpha)$ (a pair $(\widehat{\Sigma},\Theta)$ when analysing a term) that satisfies the judgements defined by the axioms and rules of Tables 6, 3 and 7.

We do not comment the clauses discussed in Section 4. The judgement $(\widehat{\Sigma},\Theta) \models_\ell M^a$, defined by the rules in Table 6, requires that $\Theta(\ell)(a)$ includes all the abstract values $\hat{v}$ associated to M^a. In the case of sensor identifiers, i^a and values v^a must be included in $\Theta(\ell)(a)$. In the case of sensor identifier also the micro-trajectory (S_i,ℓ) must be included in $T(a)$. The rule for analysing compound terms requires that the components are in turn analysed. The penultimate rule deals with the application of an r-ary encryption. To do that (i) it analyses each term $M_i^{a_i}$, and (ii) for each r-tuple of values $(\hat{v}_1,\cdots,\hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$, it requires that the abstract structured value $\{\hat{v}_1,\cdots,\hat{v}_r\}^a_{k_0}$, cut at depth d, belongs to $\Theta(\ell)(a)$. The special abstract value $\top^a$ will end up in $\Theta(\ell)(a)$ if the depth of the term exceeds d. The last rule is for the application of an r-ary function f. Also in this case, (i) it analyses each term $M_i^{a_i}$, and (ii) for all r-tuples of values $(\hat{v}_1,\cdots,\hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$, it requires that the composed abstract value $f(\hat{v}_1,\cdots,\hat{v}_r)^a$ belongs to $\Theta(\ell)(a)$.

The judgements for nodes with the form $(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models N$ are defined by the rules in Table 7. The rules for the *inactive node* and for *parallel composition* are standard. The rule for a single node $\ell : [B]$ requires that its internal components B are in turn analysed; in this case we the use rules with judgements $(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B$, where ℓ is the label of the enclosing node. The rule con-

necting actual stores Σ with abstract ones $\widehat{\Sigma}$ requires the locations of sensors to contain the corresponding abstract values. The rule for sensors is trivial, because we are only interested in the users of their values. The rules for processes

$$\frac{}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models \mathsf{0}} \qquad \frac{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models \ell : [B]} \qquad \frac{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models N_1 \wedge (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models N_2}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models N_1 \mid N_2}$$

$$\frac{\forall\, i \in \mathcal{I}_\ell . i^\ell \in \hat{\Sigma}_\ell(i)}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell \Sigma} \qquad \frac{}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell S} \qquad \frac{}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell A}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma},\Theta) \models_\ell M^a \ \wedge\ \bigwedge_{i=1}^{j}\ (\widehat{\Sigma},\Theta) \models_\ell M_i^{a_i} \ \wedge \\ \forall \{\hat{v}_1,\cdots,\hat{v}_r\}_{k_0}^b \in \Theta(\ell)(a) \Rightarrow \left(\bigwedge_{i=j+1}^{r}\ \hat{v}_i \in \hat{\Sigma}_\ell(x_i)\ \wedge\ (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell P\right)\end{array}}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell \mathsf{decrypt}\ M^a\ \mathsf{as}\ \{M_1^{a_1},\cdots,M_j^{a_j}; x_{j+1}^{a_{j+1}},\cdots,x_r^{a_r}\}_{k_0}\ \mathsf{in}\ P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma},\Theta) \models_\ell M^a \wedge \\ \forall\, \hat{v} \in \Theta(\ell)(a)\ \Rightarrow \hat{v} \in \hat{\Sigma}_\ell(x)\ \wedge (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell P\end{array}}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell x^{a_x} := M^a . P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma},\Theta) \models_\ell M^a \wedge \\ (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell P_1\ \wedge\ (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell P_2\end{array}}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell M^a ? P_1 : P_2} \qquad \frac{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B_1 \wedge (\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B_2}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B_1 \| B_2}$$

$$\frac{}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell \mathsf{0}} \qquad \frac{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell P}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell \mu h.\ P} \qquad \frac{}{(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell h}$$

Table 7. Analysis of nodes $(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models N$, and of node components $(\widehat{\Sigma},\kappa,\Theta,T,\alpha) \models_\ell B$, without the rules introduced in Table 3.

require to analyse the immediate sub-processes. The rule for *decryption* is similar to the one for communication: it also requires that the keys coincide. The rule for *assignment* requires that all the values $\hat{v}$ in the estimate $\Theta(\ell)(a)$ for M^a belong to $\widehat{\Sigma}_\ell(x)$. The rules for the *inactive process*, for *parallel composition*, and for *iteration* are standard (we assume that each iteration variable h is uniquely bound to the body P).